Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
Electronic Notes in Theoretical Computer Science 192 (2007) 77–92
www.elsevier.com/locate/entcs

An Operational Semantics for Shared Messaging Communication

Astrid Kiehn

Department of Computer Science and Engineering
Indian Institute of Technology Delhi
New Delhi 110016, India

Abstract

Shared Messaging Communication (SMC) has been introduced in [9] as a model of communication which reduces communication costs (both in terms of communication latency and memory usage) by allowing tasks to communicate data through special shared memory regions. Sending a reference to an otherwise inaccessible memory regions rather than the data itself, the model combines the advantages of message passing and shared memories. Experimental results have shown that SMC in case of large data payloads clearly outperforms the classical message passing.
In this paper we give a formal operational semantics to SMC exhibiting unambiguously the effect of executing an SMC command on local and shared memories. Based on this semantics we show that any program using message passing can be proved to be weakly bisimilar to one based on SMC and that with respect to communication costs the latter is amortised cheaper, [7].

Keywords: Shared messaging communication, operational semantics, cost evaluation, message-passing.

1 Introduction

To achieve high performance, modern computer applications are executed on networks of (multi)processors. Those with a high data rate like digital signal processing are most efficiently implemented on micro-architectures employing shared memory as a means of interprocess communication. However, shared memory programming has to deal explicitly with correct data access and data integrity and the negligence of these cause faulty computations. Architectures based on message-passing prevent such errors by barring shared address space and by their clear separation of computation and communication. But message-passing has the drawback of high data latency and redundancy of data transfer.

Shared messaging communication (SMC) aims at combining the advantages of message passing while utilizing the availability of shared memory to reduce the cost of communication. In an SMC-architecture communication is performed via references (called *tokens*) to shared memory which are provided by the network and which ensure mutually exclusive access. To communicate, a process asks for a

1571-0661
doi:10.1016/j.entcs.2007.08.021

token (by `get_unused_memory`), writes its data to the granted memory region and then sends the token to the target node (`send_token`) by message-passing. On the receiving side, the process receives the token (`receive_token`), reads or writes on the assigned memory region and then either sends the token to another node or releases it (`usage_over`). In general, when the volume of data transferred is very high, compared to the expense of granting, releasing, sending or receiving a token, the overall performance of shared-messaging communication can be significantly better than direct data communication.

The SMC-model of communication [1] has been introduced in [9]. It describes the semantics of the SMC communication primitives by the pre- and postconditions in Figure 1 and reports experimental results showing that SMC outperforms message-passing if large data items are to be communicated.

SMC supports a system level design where certain features of a class of implementations has been abstracted to a level which is amenable to a formal verification (cf. [6]). Employing SMC one goes beyond the purely qualitative behaviour description. A current program using SMC as model of communication does not only specify the (qualitative) computation with respect to functionality but also makes (quantitative) assumptions by taking the decision that the underlying mode of data transfer should rely on shared memory.

The contribution of this paper is a formal operational semantics for a class of concurrent programs of which the set of constituting sequential components is partitioned into regions and where communication within a region is performed via SMC. The partitioning into regions mirrors the (quantitative) assumption of an expected efficient implementation. The use of SMC as a model of communication guarantees that conceptually it can be considered as message passing (which is formally proved in this paper).

The semantics is based on the clear separation between local and shared memory and shows how tokens are administered and dealt with when sent to or received by other processes of the same region. Based on this semantics we are able to prove the claim made in [9] that the new SMC model of computation does not introduce non-determinism due to the shared memory region allocation by the underlying network. In particular, we show that any program based on message-passing can be rewritten to one based on SMC such that the two programs are observation equivalent ([8]). Moreover, we show that if communication costs are taken into account the message-passing program is (amortised) more expensive than the SMC-based counterpart underpinning the experimental results of [9]. The latter comparison is based on the notion of amortised bisimilarity introduced in [7]. Other bisimulation-based behavioural preorders like the efficiency preorder of [10] and [2] turned out not to be capable of expressing this relation adequately.

[1] The variant for broadcasting given in [9] is not dealt with in this paper.

Access Method	Pre-conditions on token_handle	Post-conditions on token_handle	Invoked by	Operation
get_unused_memory (token_handle)	should not hold any token	holds a token with RW privilege	sender	blocks if an unused memory is not available, else a token to an unused memory region is created
send_token (token_handle)	should hold a token with RW privilege	does not hold any token	sender	blocks if the token buffer is full, else token is transferred from the token_handle to token_buffer
receive_token (token_handle)	should not hold any token	holds a token with RW privilege	receiver	blocks if the token buffer is empty, else any earliest token from token buffer is transferred to token_handle
usage_over (token_handle)	should hold a token with RW privilege	does not hold any token	receiver	token is destroyed and the associated memory region is marked unused

Fig. 1. Semantics of SMC-commands as given in [9].

2 Shared Messaging Communication (SMC)

An SMC-program consists of a finite set of sequential programs partitioned into so-called regions. The processes of a region – which in analogy to networks we call nodes – communicate with each other using the SMC communication primitives. Intuitively, a region comprises those nodes which in the final mapping to a network should be located such that they can efficiently communicate via shared memories. We right here make the assumption that we deal with one universal region, only, to keep the notational overhead as small as possible. The general case will be discussed in the conclusions.

To describe the behaviour of a single node we use the core commands of an imperative language and intersperse them with the SMC communication primitives. So an SMC-program (assuming one global region) is a set of sequential programs $p_1, \ldots, p_n$ derived from the following grammar:

$$p_i :== \mathbf{skip} \mid l := a \mid p_i; p_i \mid \mathbf{if}\ b\ \mathbf{then}\ p_i\ \mathbf{else}\ p_i \mid \mathbf{while}\ b\ \mathbf{do}\ p_i \mid$$
$$\mathbf{gum}_i(t,k) \mid \mathbf{uo}_i(t,k) \mid \mathbf{st}_i^j(t,k) \mid \mathbf{rt}_j^i(t,k) \mid \mathbf{cps}_i(t,a) \mid \mathbf{csm}_i(t,l)$$

Each node operates on its private memory space while interprocess communication is based on SMC which affects the shared memory of the entire region. Accordingly, the operational semantics is given in two layers. First we define the semantics of single nodes and based on that the semantics of regions is given.

2.1 The Operational Semantics of Nodes

The semantics of the commands in the first line of the grammar for SMC-programs is completely standard (see e.g. [11]) and we assume familiarity with its presentation by a set of rules. In notation we use a for an arithmetic expression, b for a boolean

expression and $l \in Loc_i$ for a location. As in our context the computation of the values of these expressions is of no interest, we adopt a big step semantics here. We assume a relation $\rightsquigarrow$ which evaluates an arithmetic expression a to a (non-specified) value v and a boolean expression b to *true* or *false* wrt a given assignment of locations σ: $a, \sigma \rightsquigarrow v$ and $b, \sigma \rightsquigarrow \mathit{true}$ or $b, \sigma \rightsquigarrow \mathit{false}$.

The state of a node is given by a pair $\langle p_i, \sigma_i \rangle$ where p_i is the program that remains to be executed and σ_i is a mapping providing the actual values of locations. We give a small step semantics here to clearly demonstrate the changes of local and shared memory with a computation step. To simplify the behaviour analysis later on we decorate transitions with the assignments or evaluation that is performed with an execution step.

$$\frac{}{\langle \mathbf{skip}, \sigma_i \rangle \xrightarrow{\mathbf{skip}} \sigma_i} \qquad \frac{a, \sigma_i \rightsquigarrow v}{\langle l := a, \sigma_i \rangle \xrightarrow{l:=v} \sigma_i\{l := v\}}$$

$$\frac{\langle p_i, \sigma_i \rangle \xrightarrow{\alpha} \langle p_i', \sigma_i' \rangle}{\langle p_i; q_i, \sigma_i \rangle \xrightarrow{\alpha} \langle p_i'; q_i, \sigma_i' \rangle} \qquad \frac{\langle p_i, \sigma_i \rangle \xrightarrow{\alpha} \sigma_i'}{\langle p_i; q_i, \sigma_i \rangle \xrightarrow{\alpha} \langle q_i, \sigma_i' \rangle}$$

$$\frac{b, \sigma_i \rightsquigarrow \mathit{true}}{\langle \mathbf{if}\ b\ \mathbf{then}\ p_i\ \mathbf{else}\ q_i, \sigma_i \rangle \xrightarrow{b=\mathit{true}} \langle p_i, \sigma_i \rangle} \qquad \frac{b, \sigma_i \rightsquigarrow \mathit{false}}{\langle \mathbf{if}\ b\ \mathbf{then}\ p_i\ \mathbf{else}\ q_i, \sigma_i \rangle \xrightarrow{b=\mathit{false}} \langle q_i, \sigma_i \rangle}$$

$$\frac{b, \sigma_i \rightsquigarrow \mathit{true}}{\langle \mathbf{while}\ b\ \mathbf{do}\ p_i, \sigma_i \rangle \xrightarrow{b=\mathit{true}} \langle p_i; \mathbf{while}\ b\ \mathbf{do}\ p_i, \sigma_i \rangle} \qquad \frac{b, \sigma_i \rightsquigarrow \mathit{false}}{\langle \mathbf{while}\ b\ \mathbf{then}\ p_i, \sigma_i \rangle \xrightarrow{b=\mathit{false}} \sigma_i}$$

Token handles form a third syntactic category and are ranged over by t. By Thdls_i we denote the token handles of node π_i. The SMC-commands intuitively have the following meaning where we assume the view of a node i:

$\mathbf{gum}_i(t)$	give unused memory (to be bound to t)
$\mathbf{uo}_i(t)$	usage over (of the memory given by t)
$\mathbf{st}_i^j(t)$	send token to node j
$\mathbf{rt}_j^i(t)$	receive token from node j
$\mathbf{cps}_i(t, a)$	compose token:
	write the value of a to the memory specified by t
$\mathbf{csm}_i(t, l)$	consume token:
	write the contents of the memory specified by t to location l.

$$\mathrm{GUM_i}\quad \frac{}{\langle \mathbf{gum}_i(t), \sigma_i\rangle \xrightarrow{\mathbf{gum}_i(m)} \sigma_i\{t := m\}} \quad (\sigma_i(t) = \bot)$$

$$\mathrm{UO_i}\quad \frac{}{\langle \mathbf{uo}_i(t), \sigma_i\rangle \xrightarrow{\mathbf{uo}_i(m)} \sigma_i\{t := \bot\}} \quad (\sigma_i(t) = m)$$

$$\mathrm{ST_i^j}\quad \frac{}{\langle \mathbf{st_i^j}(t), \sigma_i\rangle \xrightarrow{\mathbf{st_i^j}(m)} \sigma_i\{t := \bot\}} \quad (\sigma_i(t) = m)$$

$$\mathrm{RT_j^i}\quad \frac{}{\langle \mathbf{rt_j^i}(t), \sigma_i\rangle \xrightarrow{\mathbf{rt_j^i}(m)} \sigma_i\{t := m\}} \quad (\sigma_i(t) = \bot)$$

$$\mathrm{CPS_i}\quad \frac{a, \sigma_i \rightsquigarrow v}{\langle \mathbf{cps}_i(t, a), \sigma_i\rangle \xrightarrow{\mathbf{cps}_i(m,v)} \sigma_i} \quad (\sigma_i(t) = m)$$

$$\mathrm{CSM_i}\quad \frac{}{\langle \mathbf{csm}_i(t, l), \sigma_i\rangle \xrightarrow{\mathbf{csm}_i(m,v,l)} \sigma_i\{l := v\}} \quad (\sigma_i(t) = m)$$

Fig. 2. Node Rules

Formally, a node is a pair [2] $\pi_i = \langle p_i, \sigma_i\rangle$ where

$$\sigma_i : (\mathit{Loc}_i \cup \mathit{Thdls}_i) \to (\mathit{Values} \cup \{\bot\} \cup \mathit{SM})$$

$\sigma_i(\mathit{Loc}_i) \subseteq \mathit{Values} \cup \{\bot\}$ and $\sigma_i(\mathit{Thdls}_i) \subseteq \mathit{SM} \cup \{\bot\}$. The symbol $\bot$ denotes undefinedness of a local location or an SMC location.

As we have left expressions and locations unspecified one may also assume concurrent assignments and by this we may assume non-scalar data structures. This modelling admittedly dilutes a crucial property of SMC, namely, that not all the data written to the shared memory by a source process will not necessarily be accessed by the target process (by **csm**). However, as the emphasis of our modelling is the administration of tokens we do not give a refined model here. The model as described in [9] assumes tokens of a predefined size (which we assume to be one).

The state of the shared memory is given by σ. It is not residing at a particular node but somewhere in the region. It will only be visible when we describe the semantics of an entire region. The remaining rules required to describe the semantics of nodes are the rules for SMC-commands, Figure 2.

Some comments on the rules:
Rule CPS$_i$: With the execution of $\mathbf{cps}_i(t, a)$ the shared memory cell m bound to t

[2] If a program has completely been executed the semantics simply yield a state σ_i. We identify this with the pair $\langle \mathit{nil}, \sigma_i\rangle$ to ensure that we can always assume the result has two components.

will be updated to $\sigma_i(a)$ but since m is not local to π this updating is visible only in the context of its region.
Rules UO_i, ST_i^j: The effect of executing either $\mathbf{uo}_i(t)$ or $\mathbf{st}_i^j(t)$ is the same, locally.
Rule CSM_i: The side condition of this rule expresses that a token handle has been assigned a shared memory location.

2.2 Regions

A *region* consists of a set of nodes $\{\pi_1, \ldots, \pi_n\}$ which share a local memory SM. For each (unidirectional) channel from a node i to a node j of the region there is a channel variable c_i^j in $C = \{c_i^j \mid i, j \in \{1, \ldots, n\}, i \neq j\}$. For simplicity, again, we assume that the channel capacity is one token item only and we do not consider channels leading outside the region. Channels also act as buffers (the token-buffers mentioned in Figure 1); if the capacity is bounded the data rate of the sending process can be forced to respect the processing time of the receiver. This is a characterizing feature of blocking networks. Assuming capacity 1 the channel can simply be described as a variable which eases the overall needed notation.

Formally, a region R is given by $R = \langle(\pi_1 \parallel \cdots \parallel \pi_n), \sigma\rangle$ where σ provides the current state of SM and the channel variables: $\sigma : SM \cup C \to \mathit{Values} \cup \{\bot\} \cup \{\mathit{available}\} \cup SM \cup \{\mathit{empty}\}$ satisfying $\sigma(SM) \subseteq \mathit{Values} \cup \{\bot\} \cup \{\mathit{available}\}$ and $\sigma(C) \subseteq SM \cup \{\mathit{empty}\}$. The set of all state mappings σ is given by Σ_{SM}.

The operational semantics is such that a memory cell m is given the value *available* by σ if m is not bound to a token handle or contained in one of the channels. For a given region with n nodes, the state mapping σ and its local counterparts σ_i should be *consistent* for all nodes i, j, $i \neq j$:

$$(a) \qquad \sigma_i(t) = m \;\text{ implies }\; \sigma(m) \neq \mathit{available}$$

$$(b) \qquad \sigma(c_i^j) = m \;\text{ implies }\; \sigma_i(t) \neq m \neq \sigma_j(t') \;\text{ and }\; \sigma(m) \neq \mathit{available}$$

$$(c)\;\; \sigma_i(t) \neq \bot \neq \sigma_j(t') \;\text{ implies }\; \sigma_i(t) \neq \sigma_j(t')$$

where t and t' are any token handles, $m \in SM$.

Consistency expresses the key property of SMC: at any state of the execution of a program a token can be bound at most to one token handle, property (c), and if it is residing in a channel, (b), it is not bound to any handle neither it is available. Obviously, initial programs – which satisfy $\sigma_i(t) = \bot$ and $\sigma(c_i^j) = \mathit{empty}$ for all token handles and channels – are consistent, and we will prove later that consistency is preserved by the operational semantics. The transition rules for SMC-commands for regions are given in Figure 3.

Some comments on the rules: Let σ' denote the updated state mapping. In $R - GUM_i$ tokem m is granted to node π_i which changes its status from being available to $\bot$. In $R - UO_i$ token m is released thus $\sigma'(m) = \mathit{available}$. In $R - ST_i^j$ the token is sent to node π_j i.e. to the conecting channel c_i^j provided the latter is empty. As we assumed channel capacity 1, non-emptyness coincides with that the channel is full. With $R - CPS_i$ the updating of m to $\sigma_i(a)$ due to the local

$$\mathrm{R-GUM_i}\quad \frac{\pi_i \xrightarrow{\mathbf{gum}_i(m)} \pi_i'}{\langle(\cdots \parallel \pi_i \parallel \cdots),\sigma\rangle \xrightarrow{\mathbf{gum}_i(m)} \langle(\cdots \parallel \pi_i' \parallel \cdots),\sigma\{m := \bot\}\rangle}\quad \left(\begin{smallmatrix}\sigma(m)=\\ available\end{smallmatrix}\right)$$

$$\mathrm{R-UO_i}\quad \frac{\pi_i \xrightarrow{\mathbf{uo}_i(m)} \pi_i'}{\langle(\cdots \parallel \pi_i \parallel \cdots),\sigma\rangle \xrightarrow{\mathbf{uo}_i(m)} \langle(\cdots \parallel \pi_i' \parallel \cdots),\sigma\{m := available\}\rangle}$$

$$\mathrm{R-ST_i^j}\quad \frac{\pi_i \xrightarrow{\mathbf{st_i^j}(m)} \pi_i'}{\langle(\cdots \parallel \pi_i \parallel \cdots),\sigma\rangle \xrightarrow{\mathbf{st_i^j}(m)} \langle(\cdots \parallel \pi_i' \parallel \cdots),\sigma\{c_i^j := m\}\rangle}\quad (\sigma(c_i^j) = empty)$$

$$\mathrm{R-RT_i^j}\quad \frac{\pi_j \xrightarrow{\mathbf{rt_i^j}(m)} \pi_j'}{\langle(\cdots \parallel \pi_j \parallel \cdots),\sigma\rangle \xrightarrow{\mathbf{rt_i^j}(m)} \langle(\cdots \parallel \pi_j' \parallel \cdots),\sigma\{c_i^j := empty\}\rangle}\quad (\sigma(c_i^j) = m)$$

$$\mathrm{R-CPS_i}\quad \frac{\pi_i \xrightarrow{\mathbf{cps}_i(m,v)} \pi_i'}{\langle(\cdots \parallel \pi_i \parallel \cdots),\sigma\rangle \xrightarrow{\mathbf{cps}_i(m,v)} \langle(\cdots \parallel \pi_i' \parallel \cdots),\sigma\{m := v\}\rangle}$$

$$\mathrm{R-CSM_i}\quad \frac{\pi_i \xrightarrow{\mathbf{csm}_i(m,v,l)} \pi_i'}{\langle(\cdots \parallel \pi_i \parallel \cdots),\sigma\rangle \xrightarrow{\mathbf{csm}_i(m,v,l)} \langle(\cdots \parallel \pi_i' \parallel \cdots),\sigma\rangle}\quad (\sigma(m) = v)$$

Fig. 3. Region Rules

execution of $\mathbf{cps}_i(t,a)$ is implemented. Finally, $R-CSM_i$ dscribes the access to contents of token m; it does not change the state mapping.

Let $PROG_{SMC}$ denote the set of all shared messaging communication programs (which may be in their initial state or have been executed for some while). Formally, $PROG_{SMC}$ is the least set containing the initial state programs which is closed under transitions. The initial state of a program is given by $\sigma(c_i^j) = empty$, $\sigma_i(l) = \bot$ and $\sigma(m) = available$ for all all c_i^j, l, m.

Proposition 2.1 *(Well-Definedness of the Operational Semantics)*

(i) *All programs in $PROG_{SMC}$ are consistent.*

(ii) *The conditions on SMC-commands in Figure 1 are satisfied by the operational semantics.*

Proof. of (i) by induction on the length of the transition sequence establishing membership in $PROG_{SMC}$.
(ii) is easily verified by inspecting the respective transition rules. □

Note that the given conditions on channels might seem stronger than required in Figure 1, however, this is only due to the fact that we assumed channels of capacity one.

3 Moving from Message Passing to SMC

For an application with a high data rate it is feasible to employ shared messaging communication in its final implementation. In this section we formally prove that any program using message-passing as communication mechanism can be rewritten as an SMC-program such that the two programs are weakly bisimilar (up to some renaming of communication actions) which shows, in particular, that the (non)deterministic structure of the MP-program is preserved by the translation. We first give a formal description of the message passing model and then relate it in terms of bisimilarity to the SMC-presentation.

3.1 The Message-Passing Model

The syntax of programs using message-passing is a straightforward variation of that of SMC-programs:

$$p_i := \mathbf{skip} \mid l := a \mid \mathbf{if}\ b\ \mathbf{then}\ p_i\ \mathbf{else}\ p_i \mid \mathbf{while}\ b\ \mathbf{do}\ p_i \mid p_i; p_i \mid \mathbf{sm}_i^j(a) \mid \mathbf{rm}_j^i(l)$$

The intuitive meaning of MP-commands is

$\mathbf{sm}_i^j(a)$	send message: send the value of a to node j,
$\mathbf{rm}_j^i(l)$	read/save message from node j at location l .

As before, a region consists of a set of nodes $\{\pi_1, \ldots, \pi_n\}$ and $C = \{c_i^j \mid i, j \in \{1, \ldots, n\}, i \neq j\}$ is the set of (unidirectional) channels between nodes. We adopt the same simplicifations as for SMC by assuming channel capacity one, no channels leading outside a region and one universal region, only. The current states of the channels are given by mapping $\sigma : C \to \mathit{Values} \cup \{\bot\} \cup \{\mathit{empty}\}$.

To describe the operational semantics the rules for SMC-commands are replaced by rules for message passing for nodes and regions, respectively.

$$\mathrm{SM}_{\mathbf{i}}^{\mathbf{j}}\ \frac{a, \sigma_i \rightsquigarrow v}{\langle \mathbf{sm}_{\mathbf{i}}^{\mathbf{j}}(a), \sigma_i \rangle \overset{\mathbf{sm}_{\mathbf{i}}^{\mathbf{j}}(v)}{\longrightarrow} \sigma_i} \qquad \mathrm{RM}_{\mathbf{j}}^{\mathbf{i}}\ \frac{}{\langle \mathbf{rm}_{\mathbf{j}}^{\mathbf{i}}(l), \sigma_i \rangle \overset{\mathbf{rm}_{\mathbf{j}}^{\mathbf{i}}(v,l)}{\longrightarrow} \sigma_i\{l := v\}}$$

$$\mathrm{R-SM}_{\mathbf{i}}^{\mathbf{j}}\ \frac{\pi_i \overset{\mathbf{sm}_{\mathbf{i}}^{\mathbf{j}}(v)}{\longrightarrow} \pi_i'}{\langle (\cdots \parallel \pi_i \parallel \cdots), \sigma \rangle \overset{\mathbf{sm}_{\mathbf{i}}^{\mathbf{j}}(v)}{\longrightarrow} \langle (\cdots \parallel \pi_i' \parallel \cdots), \sigma\{c_i^j := v\} \rangle}\ (\sigma(c_i^j) = \mathit{empty})$$

$$\mathrm{R-RM}_{\mathbf{i}}^{\mathbf{j}}\ \frac{\pi_j \overset{\mathbf{rm}_{\mathbf{i}}^{\mathbf{j}}(v,l)}{\longrightarrow} \pi_j'}{\langle (\cdots \parallel \pi_j \parallel \cdots), \sigma \rangle \overset{\mathbf{rm}_{\mathbf{i}}^{\mathbf{j}}(v,l)}{\longrightarrow} \langle (\cdots \parallel \pi_j' \parallel \cdots), \sigma\{c_i^j := \mathit{empty}\} \rangle}\ \left(\begin{smallmatrix} v=\sigma(c_i^j), \\ \neq \mathit{empty} \end{smallmatrix}\right)$$

In accordance with the definitions for SMC the set $PROG_{MP}$ denotes the least set containing the initial state MP-programs which is closed under transitions. Initially, $\sigma_i(c_i^j) = \mathit{empty}$ for all j.

3.2 Weak Bisimilarity of MP- and SMC-programs

The translation of $PROG_{MP}$ into $PROG_{SMC}$ is straightforward. We replace each command

$$\mathbf{sm}_i^j(a) \quad \text{by} \quad \mathbf{gum}_i(t); \mathbf{cps}_i(t,a); \mathbf{st}_i^j(t) \text{ and}$$
$$\mathbf{rm}_i^j(l) \quad \text{by} \quad \mathbf{rt}_i^j(t); \mathbf{csm}_j(t,l); \mathbf{uo}_j(t)$$

where for each node just one token handle is used. This substitution is described by the mapping T which we later on generalize to a relation between states of MP- and SMC-regions. As stated above, an MP-program will be related to their SMC-counterpart by weak bisimilarity. So far we have only considered so-called strong transitions opposed to weak transitions which are transitions preceded and followed by an arbitrary number of internal actions. In our setting internal actions are those concerning the administration of token handles, only. Let Act denote the set of all transition labels occurring in Section 1 and

$$\begin{aligned} Act_{\,adm} &= \{\mathbf{gum}_i(t), \mathbf{st}_i^j(t), \mathbf{rt}_i^j(t), \mathbf{uo}_j(t) \mid i,j \leq n, i \neq j\}, \\ Act_{\,SMC} &= Act_{\,adm} \cup \{\mathbf{cps}_i(m,a), \mathbf{csm}_i(v,l) \mid i \leq n, m, a, v, l \text{ arbitrary}\} \end{aligned}$$

For $\beta \in Act \cup Act_{\,SMC} - Act_{\,adm}$ weak transitions are given by

$$\begin{aligned} p \overset{\beta}{\Longrightarrow} p' &\quad \text{if } p \Rightarrow \overset{\beta}{\longrightarrow} \Rightarrow p' \text{ where} \\ p \Rightarrow p' &\quad \text{if } p \rightarrow^* p' \text{ and} \\ p \rightarrow p' &\quad \text{if there is } \alpha \in Act_{\,adm} \text{ such that } p \xrightarrow{\alpha} p'. \end{aligned}$$

For the message-passing model, we define

$$Act_{\,MP} = \{\mathbf{sm}_i^j(v), \mathbf{rm}_i^j(v,l) \mid i,j \leq n, i \neq j, v, l \text{ arbitrary}\}$$

The substitution T described above is extended to relation $\mathbf{T}$ in Figure 4. It associates MP-programs with computationally equivalent programs based on SMC. The relation is not one-to-one as the memory cell for a particular token handle is chosen arbitrarily. Note, that by $\mathbf{T}$ all $m \in SM$ which are in use (that is $\tilde{\sigma}(m) \neq \mathit{available}$) represent a message in one of the channels. This is due to the fact that $\mathbf{sm}_i^j$ and $\mathbf{rm}_i^j$ are atomic actions. States corresponding to the "halfway execution" of such an action on the SMC side will be added later.

Restricted to initial MP-programs, $\mathbf{T}$ is a function and we use $\mathbf{T}(\Pi_{mp})$ to denote the unique SMC-program. The main result of this section is that an

$$\langle(\pi_1, \| \ldots \| \pi_n), \sigma\rangle \quad \mathbf{T} \quad \langle(T(\pi_1), \| \ldots \| T(\pi_n)), \tilde{\sigma}\rangle$$

where
$\pi_i = \langle p_i, \sigma_i\rangle$, $T(\pi_i) = \langle T(p_i), T(\sigma_i)\rangle$,
$T(\sigma_i)(l) = \sigma_i(l)$ for $l \in Loc_i$ and $T(\sigma_i)(t) = \perp$,

$$\tilde{\sigma}(c_i^j) = \begin{cases} \mathit{empty} & \text{if } \sigma(c_i^j) = \mathit{empty} \\ \text{some } m \text{ such that } \tilde{\sigma}(m) = \sigma(c_i^j) & \text{otherwise} \end{cases}$$

$\tilde{\sigma}, T(\sigma_1), \ldots, T(\sigma_n)$ are consistent
and $|\{m \mid \tilde{\sigma}(m) \neq \mathit{available}\}| = |\{c_i^j \mid \sigma(c_i^j) \neq \mathit{empty}\}|$.

Fig. 4. relation $\mathbf{T} \subseteq PROG_{\mathrm{MP}} \times PROG_{SMC}$

initial program $\Pi_{mp} \in PROG_{\mathrm{MP}}$ is weakly bisimilar to $T(\Pi_{mp})$. Actually, we establish a stronger result namely that Π_{mp} and $T(\Pi_{mp})$ are in the *efficiency preorder*-relation $\preceq$, [2] (adapted to our setting). Intuitively, this means – apart from their weak bisimilarity – that the message-passing model is more efficient than the SMC-model while the experimental results suggest the converse. However, this is not a contradiction as we neglected the cost associated with sending big data items over channels.

Proposition 3.1 *(Efficiency relation between MP- and SMC-programs)*
Let $\Pi_{MP} \in PROG_{\mathrm{MP}}$, $\Pi_{SMC} \in PROG_{SMC}$.
Then $\Pi_{MP} \preceq \Pi_{SMC}$, Π_{MP} *is more efficient than* Π_{SMC}, *if there is a relation* $R \subseteq PROG_{\mathrm{MP}} \times PROG_{SMC}$ *such that for all* $(\Pi_{mp}, \Pi_{smc}) \in R$

(*i*) *if* $\Pi_{mp} \xrightarrow{\gamma} \Pi'_{mp}, \gamma \in Act$ *then* $\exists \Pi'_{smc} : \Pi_{smc} \stackrel{\gamma}{\Longrightarrow} \Pi'_{smc}$ *and* $(\Pi'_{mp}, \Pi'_{smc}) \in R$,

(*ii*) *if* $\Pi_{mp} \xrightarrow{\mathbf{sm}_i^j(v)} \Pi'_{mp}$ *then* $\exists \Pi'_{smc}, m : \Pi_{smc} \stackrel{\mathbf{cps}_i(m,v)}{\Longrightarrow} \Pi'_{smc}$ *and* $(\Pi'_{mp}, \Pi'_{smc}) \in R$,

(*iii*) *if* $\Pi_{mp} \xrightarrow{\mathbf{rm}_i^j(v,l)} \Pi'_{mp}$ *then* $\exists \Pi'_{smc}, m : \Pi_{smc} \stackrel{\mathbf{csm}_i(m,v,l)}{\Longrightarrow} \Pi'_{smc}$ *and* $(\Pi'_{mp}, \Pi'_{smc}) \in R$,

(*iv*) *if* $\Pi_{smc} \longrightarrow \Pi'_{smc}$ *then* $(\Pi_{mp}, \Pi'_{smc}) \in R$,

(*v*) *if* $\Pi_{smc} \xrightarrow{\gamma} \Pi'_{smc}, \gamma \in Act$ *then* $\exists \Pi'_{mp} : \Pi_{mp} \xrightarrow{\gamma} \Pi'_{mp}$ *and* $(\Pi'_{mp}, \Pi'_{smc}) \in R$.

(*vi*) *if* $\Pi_{smc} \xrightarrow{\mathbf{cps}_i(m,v)} \Pi'_{smc}$ *then* $\exists \Pi'_{mp} : \Pi_{mp} \xrightarrow{\mathbf{sm}_i^j(v)} \Pi'_{mp}$ *and* $(\Pi'_{mp}, \Pi'_{smc}) \in R$.

(*vii*) *if* $\Pi_{smc} \xrightarrow{\mathbf{csm}_i(m,v,l)} \Pi'_{smc}$ *then* $\exists \Pi'_{mp} : \Pi_{mp} \xrightarrow{\mathbf{rm}_i^j(v,l)} \Pi'_{mp}$ *and* $(\Pi'_{mp}, \Pi'_{smc}) \in R$.

and (Π_{MP}, Π_{SMC}) *is contained in* R.

Proof. follows directly from the definition of the efficiency preorder given in [2] where we match *sm*-actions with *cps*-actions and, respectively, *rm*-actions with

csm-actions. Actually, what we consider here is an instantiation of a ρ-σ-preorder as defined in [1]. □

By a tedious case analysis which is outlined in the next section one can establish the preorder result for a MP-program and its representation as an SMC-program. As, finally, our aim is to prove that the SMC-program is more cost-efficient than the original MP-program, and the efficiency preorder suggests the opposite result, we do not phrase it as a theorem. However, the following straightforward corollary shows the correctness of our translation.

Theorem 3.2 *Let* $\Pi = \langle (p_1 \parallel \ldots \parallel p_n), \sigma \rangle$ *be a message passing program and SM a shared memory of size at least* n*. Then* Π *is weakly bisimilar to its SMC-translation* $\mathbf{T}(\Pi)$ *where internal transitions are only those concerning the administration of token handles.*

3.2.1 Proof of the Theorem

We define a relation R which satisfies the seven conditions of Proposition 3.1. This proves $\Pi_{mp} \preceq \Pi_{smc}$. For principal reasons, every pair of processes which satisfy the efficiency preorder $\preceq$ is weakly bisimilar, see [2]. So this proves the theorem. For R we verify case (ii) in detail. From a respective analysis for the other items of Proposition 3.1 it follows $\Pi_{mp} \preceq \Pi_{smc}$.

We define R to be the least set containing $\mathbf{T}$ which is closed under the following conditions:

$$\text{If } (\Pi_{mp}\,,\,\Pi_{smc}) \in R \text{ and } \Pi_{mp} = \langle (\ldots \parallel \langle \mathbf{sm}_i^j(a); p_i, \sigma_i \rangle \parallel \ldots), \sigma \rangle,$$
$$\Pi_{smc} = \langle (\ldots \parallel \langle T(\mathbf{sm}_i^j(a)); T(p_i), T(\sigma_i) \rangle \parallel \ldots), \tilde{\sigma} \rangle$$

then

$$(\Pi_{mp}, \langle (\ldots \parallel \langle \mathbf{cps}_i(t,a); \mathbf{st}_i^j(t); T(p_i), T(\sigma_i)\{t := m\} \rangle \parallel \ldots), \tilde{\sigma}\{m := \bot\} \rangle) \in R,$$
$$(\Pi'_{mp}, \langle (\ldots \parallel \langle \mathbf{st}_i^j(t); T(p_i), T(\sigma_i)\{t := m\} \rangle \parallel \ldots), \tilde{\sigma}\{m := \sigma_i(a)\} \rangle) \in R$$

where $\Pi'_{mp} = \langle (\ldots \parallel \langle p_i, \sigma_i \rangle \parallel \ldots), \sigma \rangle$ and $\tilde{\sigma}(m) = \mathit{available}$.

$$\text{If } (\Pi_{mp}\,,\,\Pi_{smc}) \in R \text{ and } \Pi_{mp} = \langle (\ldots \parallel \langle \mathbf{rm}_i^j(l); p_j, \sigma_j \rangle \parallel \ldots), \sigma \rangle,$$
$$\Pi_{smc} = \langle (\ldots \parallel \langle T(\mathbf{rm}_i^j(l)); T(p_j), T(\sigma_j) \rangle \parallel \ldots), \tilde{\sigma} \rangle$$

then

$$(\Pi_{mp}, \langle (\ldots \parallel \langle \mathbf{csm}_j(t,l); \mathbf{uo}_j(t); T(p_j), T(\sigma_j)\{t := m\} \rangle \parallel \ldots), \tilde{\sigma}\{c_i^j := \mathit{empty}\} \rangle) \in R,$$
$$(\Pi'_{mp}, \langle (\ldots \parallel \langle \mathbf{uo}_j(t); T(p_j), T(\sigma_j)\{l := \sigma(m)\} \rangle \parallel \ldots), \tilde{\sigma}\{c_i^j := \mathit{empty}\} \rangle) \in R$$

where $\Pi'_{mp} = \langle (\ldots \parallel \langle p_j, \sigma_j \rangle \parallel \ldots), \sigma \rangle$ and $\tilde{\sigma}(c_i^j) = m$.

Lemma 3.3 *Let $\langle p_i, \sigma_i\rangle \xrightarrow{\mathbf{sm}_i^j(v)} \langle p_i', \sigma_i'\rangle$. Then there are processes q_i^1, q_i^2, q_i^3 and local state mapping σ_i^1, σ_i^2 and σ_i^3 such that*

$$\langle T(p_i), T(\sigma_i)\rangle \xrightarrow{\mathbf{gum}_i(m)} \langle q_i^1, \sigma_i^1\rangle \xrightarrow{\mathbf{cps}_i(m,v)} \langle q_i^2, \sigma_i\rangle \xrightarrow{\mathbf{st}_i^j(m)} \langle q_i^3, \sigma_i^3\rangle$$

and $q_i^3 = T(p_i')$ and $\sigma_i^3 = T(\sigma_i) = T(\sigma_i')$ and m is some shared memory cell.

Proof. We show the lemma for $p_i = \mathbf{sm}_i^j(a)$. The general case is easily derived.

As $\langle \mathbf{sm}_i^j(a), \sigma_i\rangle \xrightarrow{\mathbf{sm}_i^j(v)} \sigma_i$ we have $\sigma_i(a) = v$.

Now, for the three transitions

1. $\langle \mathbf{gum}_i(t); \mathbf{cps}_i(t,a); \mathbf{st}_i^j(t), T(\sigma_i)\rangle \xrightarrow{\mathbf{gum}_i(m)} \langle \mathbf{cps}_i(t,a); \mathbf{st}_i^j(t), \sigma_i^1\rangle$
2. $\langle \mathbf{cps}_i(t,a); \mathbf{st}_i^j(t), \sigma_i^1\rangle \xrightarrow{\mathbf{cps}_i(m,v)} \langle \mathbf{st}_i^j(t), \sigma_i^2\rangle$
3. $\langle \mathbf{st}_i^j(t), \sigma_i^2\rangle \xrightarrow{\mathbf{st}_i^j(m)} \sigma_i^3.$

it is easily proved by inspecting the corresponding transition rules that $\sigma_i^1 = T(\sigma_i)\{t := m\}$, $\sigma_i^2 = \sigma_i^1$ and $\sigma_i^3 = \sigma_i^2\{t := \bot\} = T(\sigma_i)\{t := \bot\} = T(\sigma_i)$. □

We show next, that this local transition sequence is also possible in the context of a region provided there is a free shared memory cell available.

Lemma 3.4 *Let $\Pi_{mp} = \langle (\ldots \parallel \pi_i \parallel \ldots), \sigma\rangle \in PROG_{MP}$ and $(\Pi_{mp}, \Pi_{smc}) \in \mathbf{T}$. If*

$$\langle (\ldots \parallel \pi_i \parallel \ldots), \sigma\rangle \xrightarrow{\mathbf{sm}_i^j(v)} \langle (\ldots \parallel \pi_i' \parallel \ldots), \sigma'\rangle$$

and $m \in SM$ is such that $\sigma(m) = available$ then

$$\begin{aligned}\Pi_{smc} = \langle (T(\pi_1) \parallel \ldots \parallel T(\pi_i) \parallel \ldots\ldots \parallel T(\pi_n)), \tilde{\sigma}\rangle \\ \xrightarrow{\mathbf{gum}_i(m)} \langle (\ldots \parallel \langle q_i^1, \sigma_i^1\rangle, \parallel \ldots), \sigma^1\rangle \\ \xrightarrow{\mathbf{cps}_i(m,v)} \langle (\ldots \parallel \langle q_i^2, \sigma_i^2\rangle, \parallel \ldots), \sigma^2\rangle \\ \xrightarrow{\mathbf{st}_i^j(m)} \langle (\ldots \parallel \langle q_i^3, \sigma_i^3\rangle, \parallel \ldots), \sigma^3\rangle\end{aligned}$$

where the q_i^j and σ_i^j are given by the previous lemma and $\sigma^3 = \tilde{\sigma}\{m := v, c_i^j := m\}$ and $T(\sigma_1), \ldots, T(\sigma_n)$ and σ^3 are consistent.

Proof. The first transition is possible by the choice of m and rule $\mathrm{R-GUM}_i$. Thus, $\sigma^1 = \tilde{\sigma}\{m := \bot\}$. The second transition follows from the previous lemma and rule $\mathrm{R-CPS}_i$ as $\sigma_i^1(t) = m$. So, $\sigma^2 = \sigma^1\{m := v\}$. The last transition can be deduced with rule $\mathrm{R-ST}_i^j$ as its side condition is satisfied for σ^2 because $\sigma(c_i^j) = empty$ – otherwise the local $\mathbf{sm}_i^j(v)$-transition had been impossible – and the definition of $\mathbf{T}$. So, $\sigma^3 = \tilde{\sigma}\{m := v, c_i^j := m\}$. □

Lemma 3.5 *Let $\Pi \in PROG_{SMC}$ such that each node uses at most one token handle. If Π has n nodes and global state mapping σ and n nodes then $n \geq |\{m \in SM \mid$*

$\sigma(m) \neq \mathit{available}\}|$.

Proof. by induction on the length of the transition sequence establishing membership in $PROG_{SMC}$. The crucial rules are $\mathrm{R-GUM}_i$ and $\mathrm{R-UO}_i$ in conjunction with the underlying node transitions. □

Proposition 3.6 *Let* $\Pi_{mp} = \langle(\ldots \parallel \pi_i \parallel \ldots), \sigma\rangle \in PROG_{MP}$ *and* $(\Pi_{mp}, \Pi_{smp}) \in \mathbf{T}$. *Let,* $|SM| \geq n$.

If $$\langle(\ldots \parallel \pi_i \parallel \ldots), \sigma\rangle \xrightarrow{\mathbf{sm}_i^j(v)} \underbrace{\langle(\ldots \parallel \pi_i' \parallel \ldots), \sigma'\rangle}_{\Pi_{mp}'}$$

then $$\Pi_{smc} = \langle(\ldots \parallel T(\pi_i) \parallel \ldots), \tilde{\sigma}\rangle \xRightarrow{\mathbf{cps}_i(m,v)} \underbrace{\langle(\ldots \parallel T(\pi') \parallel \ldots), \tilde{\sigma}'\rangle}_{\Pi_{smc}{}'}$$

and $$(\Pi_{mp}', \tilde{\Pi}_{mp}') \in \mathbf{T} \text{ where } \quad \tilde{\sigma}' = \tilde{\sigma}\{m := v, c_i^j := m\}.$$

Proof. By Lemma 3.5 and Lemma 3.4. □

4 Cost Analysis

The discussion in the previous section has shown that a comparison based on matching actions, only, does not capture the different performance of systems but rather suggests a relation which on intuitive grounds one would reject. However, if one assigns costs to actions and matches actions while keeping track of the current cost balance, the quantitative performance can also be observed. To this end, amortised bisimilarity has been introduced in [7]. In short, it combines bisimilarity with an quantitative cost evaluation.

The main idea is to consider actions together with their costs and to modify bisimulation equivalence in such a way that actions are matched with "functionally equivalent" actions. The difference in their costs adds to the credit which is accumulated during the mutual simulation. For a system p to be considered less expensive than another system q, the amortised bisimulation containing (p, q) should have nonnegative credit everywhere. We are interested in applications where some additinal internal activity – like the administration of tokens in SMC – increases the performance of the system in some way. It is weak amortised bisimilarity that is of interest in this setting as the internal activity corresponds to invisible τ-actions. So we allow a visible action to be simulated by an action which is preceded and followed by a sequence of costly actions which are functionally equivalent to τ. Functional equivalence is given by relation ρ. In general, it is only a small set of actions for which ρ does not reduce to identity and costs are considered different from 0. A weak transition is a sequence $p \overset{uav}{\Longrightarrow} p'$ where a is a visible action and u and v are actions which are functionally equivalent to τ, that is $u = b_1 \ldots b_n$, $v = d_1 \ldots d_m$ and $b_i \rho \tau$, $d_j \rho \tau$ for all i, j. In short, $u \rho \varepsilon$ and $v \rho \varepsilon$. Every action a is equipped with a nonnegative cost $c_a \in \mathbb{N}$. For words $u = u_1 \ldots u_n$ we have $c_u = c_{u_1} + \cdots + c_{u_n}$.

Definition 4.1 Let $\langle \mathbf{P}, \mathcal{A}_\tau, \longrightarrow\rangle$ be a labelled transition system with transition labels in $\mathcal{A}_\tau = \mathcal{A} \cup \{\tau\}$, $\rho \subseteq \mathcal{A}_\tau \times \mathcal{A}_\tau$ and every action $a \in \mathcal{A}_\tau$ carries costs c_a (by

definition $c_\tau = 0$). A family $(R_i)_{i\in\mathbb{N}}$ of binary relations over $\mathbf{P}$ is a weak *amortised ρ-bisimulation*, if for all $i \in \mathbb{N}$, $(p,q) \in R_i$:

(i) $p \xrightarrow{a} p'$ implies $\exists q', b,\, u, v\ [a\rho b,\, \varepsilon\rho\, uv,\, q \overset{u\hat{b}v}{\Longrightarrow} q'$ and $(p',q') \in R_{i+c_{u\hat{b}v}-c_a}]$,

(ii) $q \xrightarrow{b} q'$ implies $\exists p', a,\, u, v\ [a\rho b,\, uv\rho\,\varepsilon, p \overset{u\hat{a}v}{\Longrightarrow} p'$ and $(p',q') \in R_{i+c_b-c_{u\hat{a}v}}]$,

where $a, b \in \mathcal{A}_\tau$ and $u, v \in \mathcal{A}^*$ and $\hat{a} = a$ if $a \neq \tau$ and $\hat{\tau} = \varepsilon$. Process p is (weakly) amortised cheaper (more cost efficient) than q up to credit i, $p \preccurlyeq_i^\rho q$, if $(p,q) \in R_i$ for some weak amortised ρ-bisimulation $(R_i)_{i\in\mathbb{N}}$.

The costly actions in our applications which are functionally equivalent to τ are those in $Act_{\,adm}$. They carry cost 1. Actions of type *cps* and *csm* are mapped to *sm* and *rm*, respectively, but to the former we assign cost 0 as they are local read and write operations opposed to the latter channel operations which we give cost 2.

One should note here that due to the simplification in the formal modelling of SMC the real benefit of SMC gets diluted. However, the reader should readily see that if large data packets are sent over the channels then the credit in the amortised bisimulation will increase significantly. We also neglect that in case of very small data items the cost of token administration might exceed the cost of simply sending the data by message passing. Naturally, the overall speedup also depends on the ratio of computation to communication. To give some real data, for the application of a JPEG encoder implemented with SMC the overall speedup has been shown 1.77 over the message-passing model, see [9].

The transition system we consider is the disjoint union of the transition systems induced by the operational semantics of MP- and SMC-programs. We define an amortised bisimulation which for each pair $(\Pi_{smc}, \Pi_{mp}) \in R_i$ satisfies

(i) if $\Pi_{smc} \xrightarrow{\alpha} \Pi'_{smc}, \alpha \in Act_{\,adm}$, then $(\Pi'_{smc}, \Pi_{mp}) \in R_{i-1}$,

(ii) if $\Pi_{smc} \xrightarrow{\gamma} \Pi'_{smc}, \gamma \in Act$ then there exist Π'_{mp} :
$\Pi_{mp} \xrightarrow{\gamma} \Pi'_{mp}$ and $(\Pi'_{smc}, \Pi'_{mp}) \in R_i$.

(iii) if $\Pi_{smc} \xrightarrow{\mathbf{cps}_i(m,v)} \Pi'_{smc}$ then there exist Π'_{mp}, j, v :
$\Pi_{mp} \xrightarrow{\mathbf{sm}_i^j(v)} \Pi'_{mp}$ and $(\Pi'_{smc}, \Pi'_{mp}) \in R_{i+2}$.

(iv) if $\Pi_{smc} \xrightarrow{\mathbf{csm}_i(m,v,l)} \Pi'_{smc}$ then there exist Π'_{mp}, j, v, l :
$\Pi_{mp} \xrightarrow{\mathbf{rm}_i^j(v,l)} \Pi'_{mp}$ and $(\Pi'_{smc}, \Pi'_{mp}) \in R_{i+2}$.

(v) if $\Pi_{mp} \xrightarrow{\gamma} \Pi'_{mp}, \gamma \in Act$ then there exist $u \in Act^*_{\,adm}, \Pi'_{smc}$:
$\Pi_{smc} \overset{u\gamma}{\Longrightarrow} \Pi'_{smc}$ and $(\Pi'_{smc}, \Pi'_{mp}) \in R_{i-c_u}$,

(vi) if $\Pi_{mp} \xrightarrow{\mathbf{sm}_i^j(v)} \Pi'_{mp}$ then there exist $u, w \in Act^*_{\,adm}, \Pi'_{smc}, m$:
$\Pi_{smc} \overset{u}{\Longrightarrow} \xrightarrow{\mathbf{cps}_i(m,v)} \overset{w}{\Longrightarrow} \Pi'_{smc}$ and $(\Pi'_{smc}, \Pi'_{mp}) \in R_{i+2-c_{uw}}$,

(*vii*) if $\Pi_{mp} \xrightarrow{\mathbf{rm}_i^j(v,l)} \Pi'_{mp}$ then there exist $u, w \in Act^*_{adm}, \Pi'_{smc}, m$:

$$\Pi_{smc} \stackrel{u}{\Longrightarrow} \xrightarrow{\mathbf{csm}_i(m,v,l)} \stackrel{w}{\Longrightarrow} \Pi'_{smc} \text{ and } (\Pi'_{smc}, \Pi'_{mp}) \in R_{i+2-c_{uw}}.$$

Note, that (i) – (iv) and (v) – (vii) directly correspond to conditions (iv) – (vii) and (i) – (iii) given in Proposition 3.1. The two groups are swapped here as we consider a reverse ordering. Since all actions in Act_{adm} are mapped to τ by ρ it suffices to verify conditions (i) – (vii) to prove that $(R_i)_{i\in\mathbb{N}}$ is an amortised bisimulation. Using relation R defined in the last section we set up the family $(R_i)_{i\in\mathbb{N}}$:

$$(\Pi_{smc}, \Pi_{mp}) \in R_i \text{ if and only if } (\Pi_{mp}, \Pi_{smc}) \in R \text{ and } i \geq D(\Pi_{smc})$$

where $D(\Pi_{smc})$ is defined as $D(\Pi_{smc}) = \sum_{i=1}^{n} D(p_i)$ and

$$D(p_i) = \begin{cases} 0 \text{ if } p_i \xrightarrow{\mathbf{cps}_i(m,v)} \text{ or } p_i \xrightarrow{\mathbf{csm}_i(m,v)} \text{ for some } m, v \\ 2 \text{ if } p_i \xrightarrow{\mathbf{st}_i^j(m)} \text{ or } p_i \xrightarrow{\mathbf{uo}_i(m)} \text{ for some } j, m \\ 1 \text{ otherwise} \end{cases}$$

$D(p_i)$ gives the maximal credit demand that is necessary to perform the initial token administration. It is easily verified that conditions (i) – (vii) indeed hold. As every initial program Π with n nodes has maximal demand n in its SMC version the following proposition is straightforward.

Theorem 4.2 *If Π_{mp} is an initial MP-program with n sequential components and Π_{smc} is its SMC-presentation that is $\Pi_{smc} = T(\Pi_{mp})$ then Π_{smc} is amortised cheaper than Π_{mp} with credit n, $|SM| \geq n$: $T(\Pi) \preccurlyeq^{\rho}_{n} \Pi$ where ρ is defined as after Def. 4.1.*

5 Conclusions

We have given an operational semantics for concurrent programs whose sequential processes are partionened into regions to facilitate an efficient implementation. The processes forming a region are assumed to communicate via shared memory in the final implementation. We have restricted the formal model to one region, only, but the set-up is easily generalized. In that case the whole system is described by $\langle \langle (\pi^1_1 \parallel \ldots \parallel \pi^1_{n_1}), \sigma^1 \rangle, \ldots, \langle (\pi^m_1 \parallel \ldots \parallel \pi^m_{n_m}), \sigma^m \rangle, C \rangle$ where each $\langle (\pi^i_1 \parallel \ldots \parallel \pi^i_{n_i}), \sigma^i \rangle$ is a region as described in Section 2. Mapping C provides the current state for each inter-region channel. The semantics of inter-region channels is defined as in the section on message-passing and the channel access operations are **sm** and **rm**.

SMC-programs support a system design methodology – called *orthogonalization of concerns* in [6] – which separates functional correctness from other, quantitative aspects. A program using message-passing as model of communication (only) can be proved functionally correct independently of any concern with respect to an efficient transfer of data. Once this has been established the question of performance can be tackled. We have shown in this paper that one may move from message-passing to

shared messaging communication without loosing functional correctness but with higher efficiency if the volume of data transferred is high. SMC may be seen as a mechanism efficiently implementing message-passing for high data-rate applications and as such it goes in line with e.g. FLASH architecture [3], ARACHNE protocol [4] or efficient implementations of value passing languages [5]. However, SMC provides a new communication model transperant to the programmer and this supports portability and reusability of software within a certain class of micro-architectures.

References

[1] S. Arun-Kumar. On bisimilarities induced by relations on actions. In *Conference on Software Engineering and Formal Methods*. IEEE, 2006.

[2] S. Arun-Kumar and M. Hennessy. An efficiency preorder for processes. *Acta Informatica*, 1(29):737–760, 1992.

[3] Kushin et. al. The Stanford FLASH multiprocessor. In *Proceedings of the 21st International Symposium on Computer Architecture*, 1994.

[4] K. G. W. Goossens. A protocol and memory manager for on-chip communication. In *IEEE International Symposium on Circuits and Systems*, 2001.

[5] A. Ingólfsdóttir and R. Pugliese. Towards verified lazy implementation of concurrent value-passing languages. In W. Brauer, editor, *Net Theory and Application*, volume 27 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1999.

[6] K. Keutzer, S. Malik, R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincentelli. System-level design: orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(12):1523–1543, 2000.

[7] A. Kiehn and S. Arun-Kumar. Amortised bisimulations. In *Proceedings of FORTE 2005*, number 3731 in Lecture Notes in Computer Science, pages 320–334. Springer–Verlag, 2005.

[8] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[9] Satya Kiran M.N.V., Jayram M.N., Pradeep Rao, and S.K. Nandy. A complexity effective communication model for behavioral modeling of signal processing applications. In *Proceedings of 40th Design Automation Conference*, 2003.

[10] F. Moller and C. Tofts. Relating processes with respect to speed. In *Proceedings of CONCUR 91*, number 527 in Lecture Notes in Computer Science, pages 424–438. Springer–Verlag, 1991.

[11] G. Winskel. *The Formal Semantics of Programming Languages*. MIT-Press, 1993.